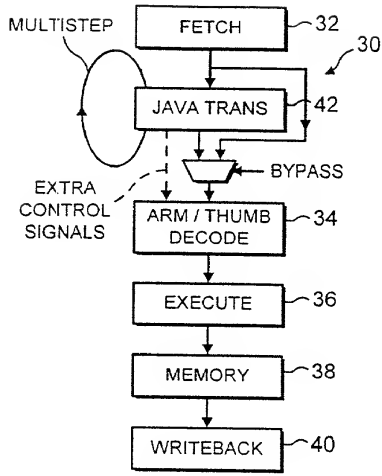
MULTISTEP
FETCH
32
30
JAVA TRANS
42
BYPASS
EXTRA
CONTROL
SIGNALS
ARM / THUMB
DECODE
34
EXECUTE
36
MEMORY
38
WRITEBACK
40

FIG. 1

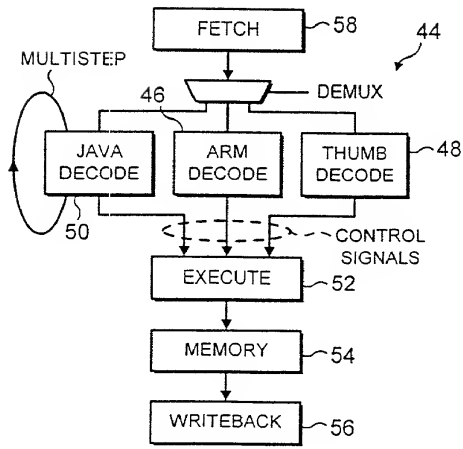
FETCH
58
44
MULTISTEP
46
DEMUX
JAVA
DECODE
ARM
DECODE
THUMB
DECODE
48
50
CONTROL
SIGNALS
EXECUTE
52
MEMORY
54
WRITEBACK
56

FIG. 2

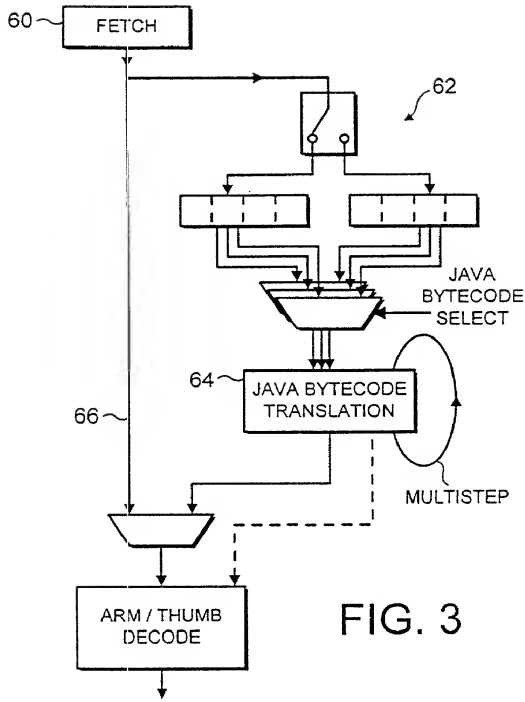
60
FETCH
62
JAVA
BYTECODE
SELECT
64
JAVA BYTECODE
TRANSLATION
66
MULTISTEP
ARM / THUMB
DECODE

FIG. 3

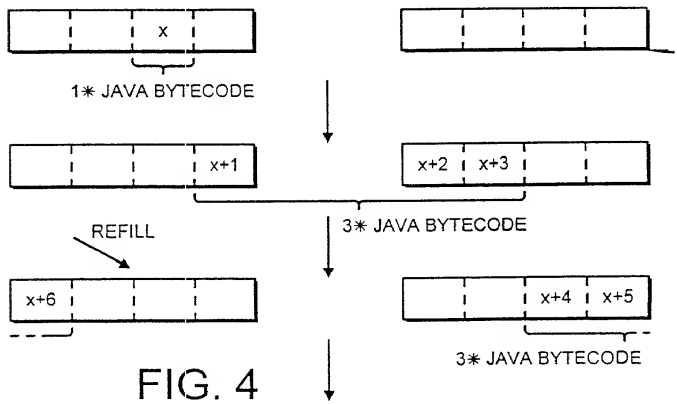
x
1* JAVA BYTECODE
x+1
x+2
x+3
3* JAVA BYTECODE
REFILL
x+6
x+4
x+5
3* JAVA BYTECODE

FIG. 4

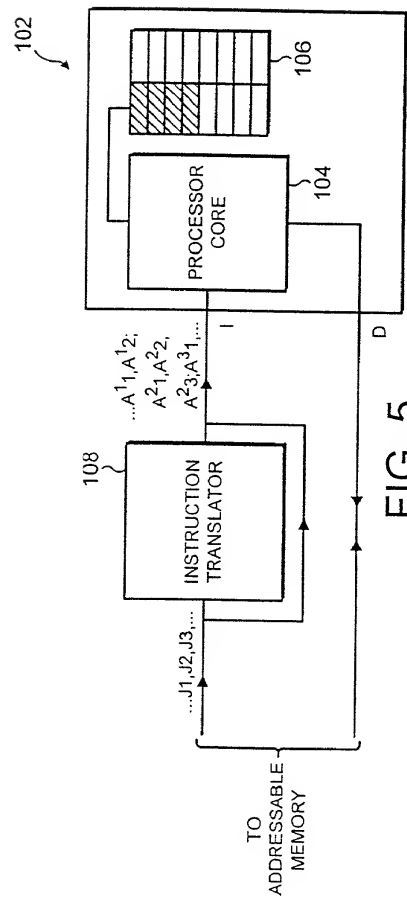
102
106
PROCESSOR CORE
104
I
D
...A1,A12,
A21,A22,
A23,A31,...
108
INSTRUCTION TRANSLATOR
...J1,J2,J3,...
TO ADDRESSABLE MEMORY

FIG. 5

JAVA INSTRUCTION	iadd (RF=2,RF>Ø)		iadd (RF=2,RF>1)	iadd (SA=-1)
ARM INSTRUCTION(S)	LDR RØ[RStack,#-4]! (POP)		LDR R3[RStack,#-4]! (POP)	ADD R3, R3, RØ
STATE	00000	00100	01000	00111
RØ	E	SOA TOS	SOA TOS	E
R1	E	E	E	E
R2	E	E	E	E
R3	E	E	SOB TOS-1	(SOA+SOB) TOS

JAVA INSTRUCTION	$lload^1$ (RF=Ø,RE>2)		$lload^2$ (RF=Ø,RE>2)	$lload^2$ (RF=Ø,RE=2)
ARM INSTRUCTIONS	LDR R1,[Rvars,#4] LDR RØ,[Rvars,#0]		STR R3[RStack],#-4 (PUSH)	LDR R3,[Rvars,#4] LDR R2,[Rvars,#0]
STATE	00000	00100	01000	10011
RØ	E	SOC TOS-1	SOC TOS-1	SOC TOS-3
R1	E	SOD TOS	SOD TOS	SOD TOS-2
R2	E	E	E	SOE TOS-1
R3	(SOA+SOB) TOS	(SOA+SOB) TOS	E	SOF TOS

FIG. 6

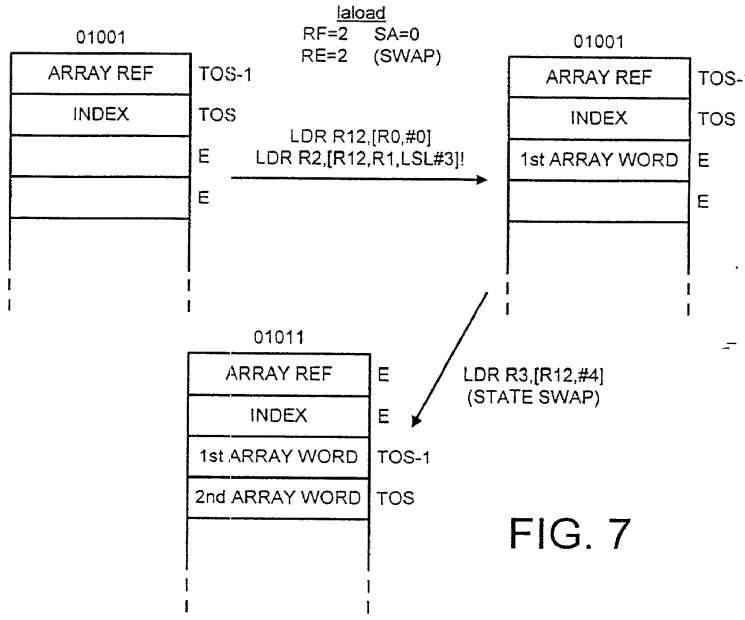
laload
RF=2 SA=0
RE=2 (SWAP)
01001
ARRAY REF
INDEX
TOS-1
TOS
E
E
LDR R12,[R0,#0]
LDR R2,[R12,R1,LSL#3]!
01001
ARRAY REF
INDEX
1st ARRAY WORD
TOS-1
TOS
E
E
01011
ARRAY REF
INDEX
1st ARRAY WORD
2nd ARRAY WORD
E
E
TOS-1
TOS
LDR R3,[R12,#4]
(STATE SWAP)

FIG. 7

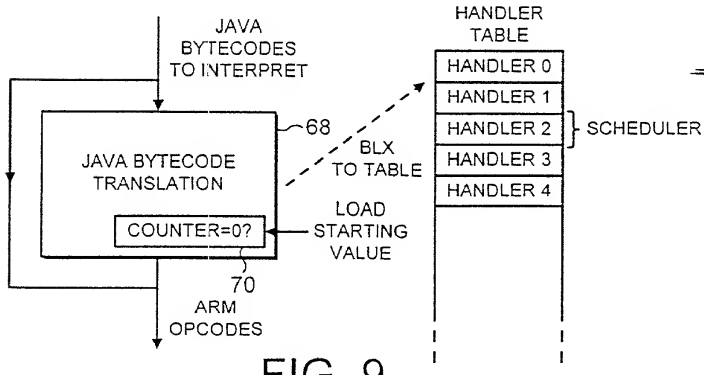
JAVA
BYTECODES
TO INTERPRET
JAVA BYTECODE
TRANSLATION
COUNTER=0?
68
70
ARM
OPCODES
BLX
TO TABLE
LOAD
STARTING
VALUE
HANDLER
TABLE
HANDLER 0
HANDLER 1
HANDLER 2
HANDLER 3
HANDLER 4
SCHEDULER

FIG. 9

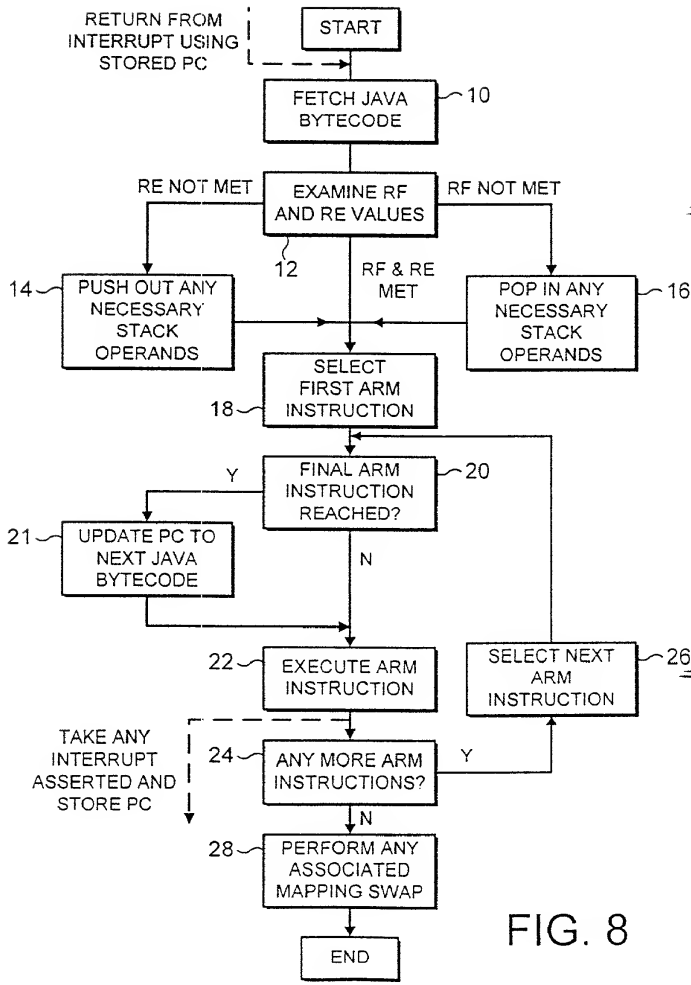
RETURN FROM INTERRUPT USING STORED PC
START
FETCH JAVA BYTECODE
10
EXAMINE RF AND RE VALUES
12
RE NOT MET
RF NOT MET
RF & RE MET
14
PUSH OUT ANY NECESSARY STACK OPERANDS
16
POP IN ANY NECESSARY STACK OPERANDS
SELECT FIRST ARM INSTRUCTION
18
FINAL ARM INSTRUCTION REACHED?
20
Y
N
21
UPDATE PC TO NEXT JAVA BYTECODE
22
EXECUTE ARM INSTRUCTION
SELECT NEXT ARM INSTRUCTION
26
TAKE ANY INTERRUPT ASSERTED AND STORE PC
24
ANY MORE ARM INSTRUCTIONS?
Y
N
28
PERFORM ANY ASSOCIATED MAPPING SWAP
END

FIG. 8

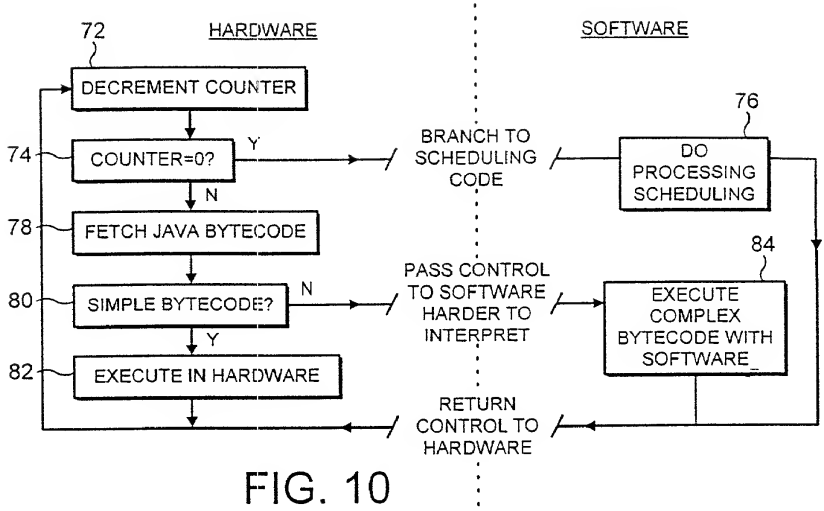
72
HARDWARE
SOFTWARE
DECREMENT COUNTER
74
COUNTER=0?
Y
N
BRANCH TO SCHEDULING CODE
76
DO PROCESSING SCHEDULING
78
FETCH JAVA BYTECODE
80
SIMPLE BYTECODE?
N
Y
PASS CONTROL TO SOFTWARE HARDER TO INTERPRET
84
EXECUTE COMPLEX BYTECODE WITH SOFTWARE
82
EXECUTE IN HARDWARE
RETURN CONTROL TO HARDWARE

FIG. 10

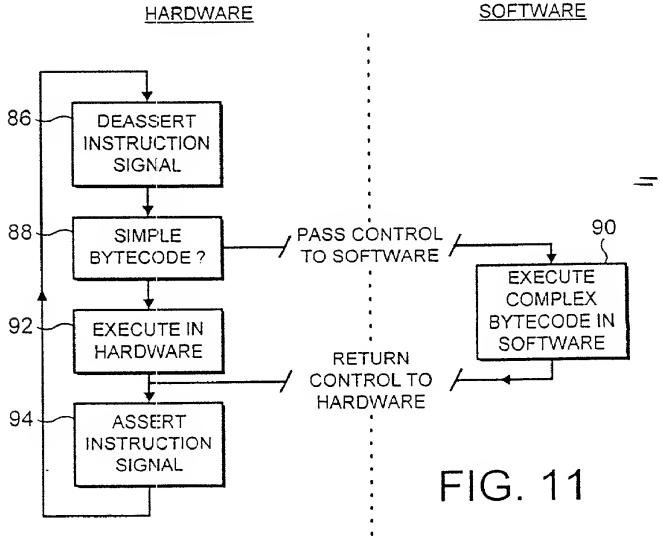
HARDWARE
SOFTWARE
86
DEASSERT INSTRUCTION SIGNAL
88
SIMPLE BYTECODE ?
PASS CONTROL TO SOFTWARE
90
EXECUTE COMPLEX BYTECODE IN SOFTWARE
92
EXECUTE IN HARDWARE
RETURN CONTROL TO HARDWARE
94
ASSERT INSTRUCTION SIGNAL

FIG. 11

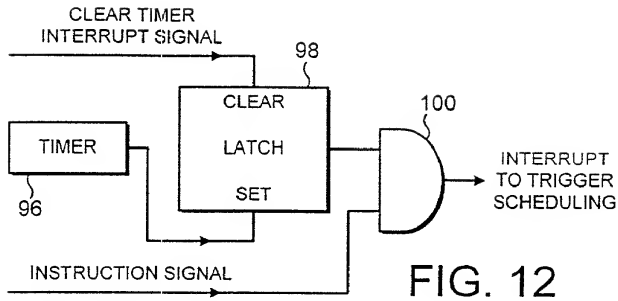
CLEAR TIMER
INTERRUPT SIGNAL
98
CLEAR
LATCH
SET
100
TIMER
96
INTERRUPT
TO TRIGGER
SCHEDULING
INSTRUCTION SIGNAL

FIG. 12

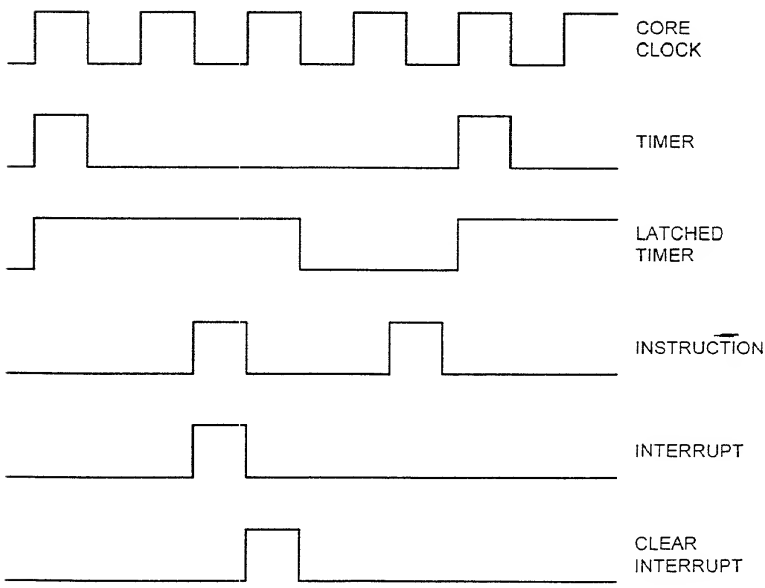
CORE
CLOCK
TIMER
LATCHED
TIMER
INSTRUCTION
INTERRUPT
CLEAR
INTERRUPT

FIG. 13

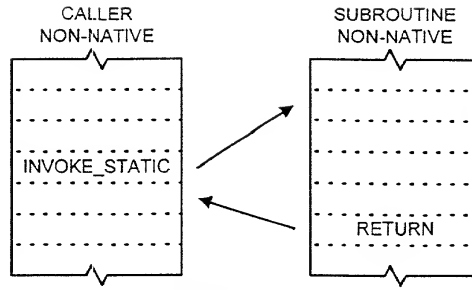

FIG. 14
PRIOR ART

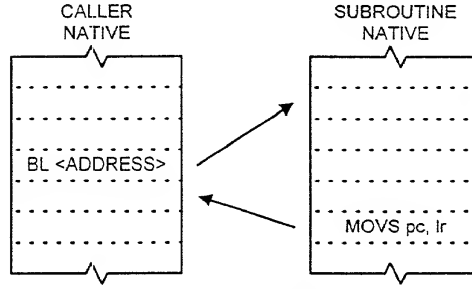

FIG. 15
PRIOR ART

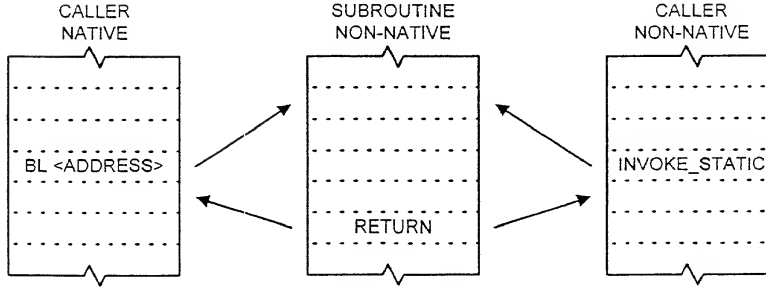

FIG. 16
PRIOR ART

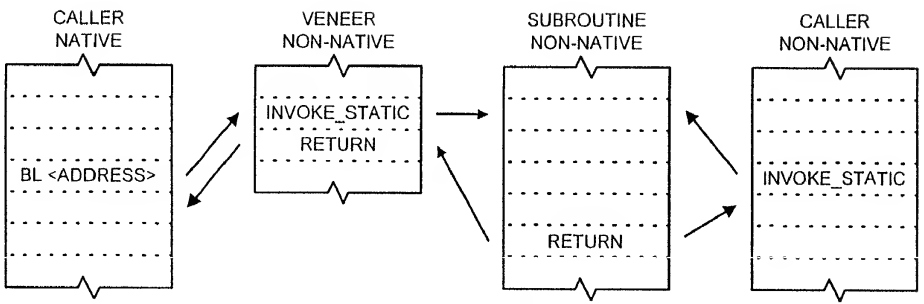

FIG. 17
PRIOR ART

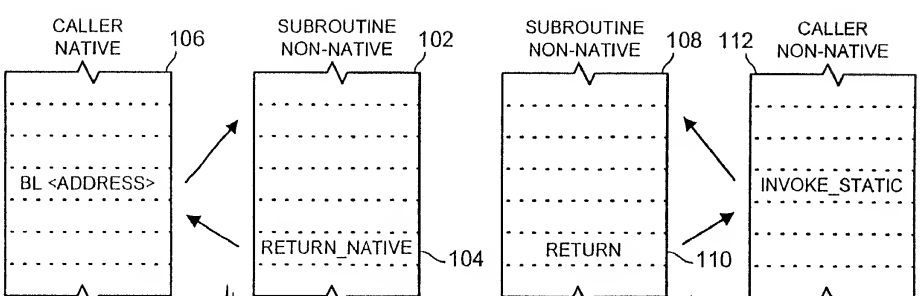

FIG. 18

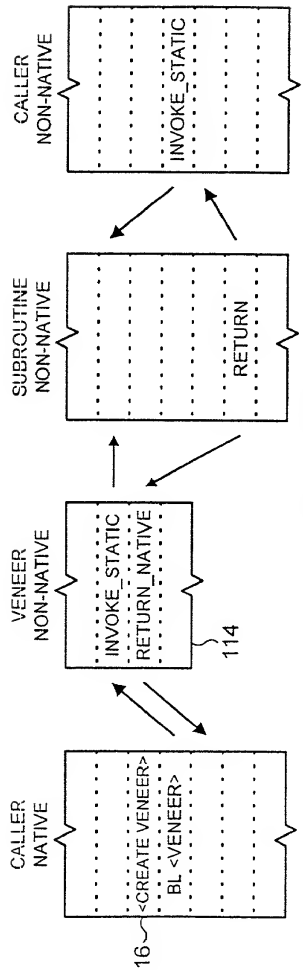
CALLER
NATIVE
<CREATE VENEER>
BL <VENEER>
116
VENEER
NON-NATIVE
INVOKE_STATIC
RETURN_NATIVE
114
SUBROUTINE
NON-NATIVE
RETURN
CALLER
NON-NATIVE
INVOKE_STATIC

FIG. 19

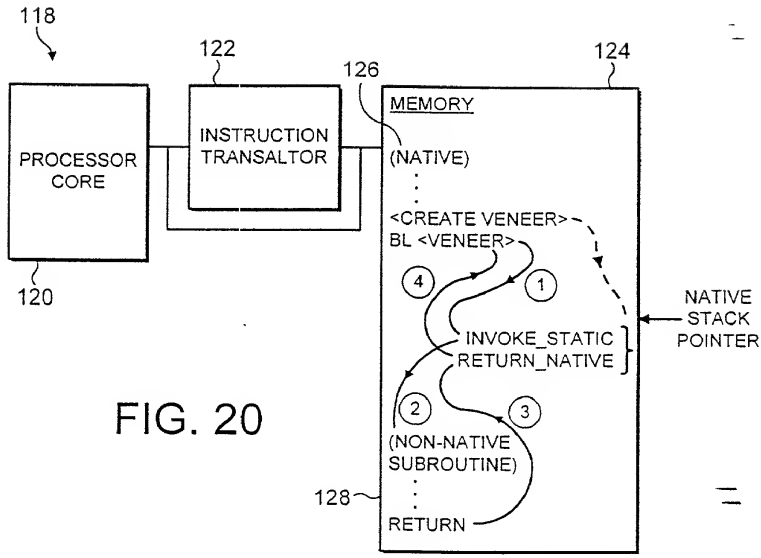
118
122
126
124
PROCESSOR CORE
INSTRUCTION TRANSALTOR
MEMORY
(NATIVE)
<CREATE VENEER>
BL <VENEER>
4
1
INVOKE_STATIC
RETURN_NATIVE
NATIVE STACK POINTER
2
3
(NON-NATIVE SUBROUTINE)
128
RETURN
120

FIG. 20

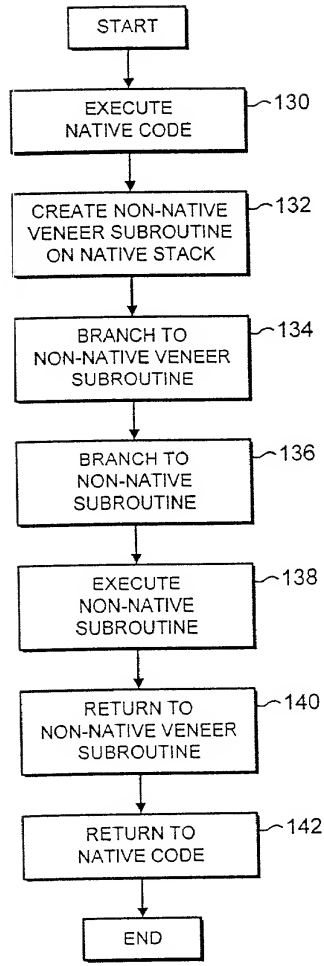
START
EXECUTE NATIVE CODE
130
CREATE NON-NATIVE VENEER SUBROUTINE ON NATIVE STACK
132
BRANCH TO NON-NATIVE VENEER SUBROUTINE
134
BRANCH TO NON-NATIVE SUBROUTINE
136
EXECUTE NON-NATIVE SUBROUTINE
138
RETURN TO NON-NATIVE VENEER SUBROUTINE
140
RETURN TO NATIVE CODE
142
END

FIG. 21